Test_Solution

Wenjie Lan

2025-03-11

```
library(dirichletprocess)
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

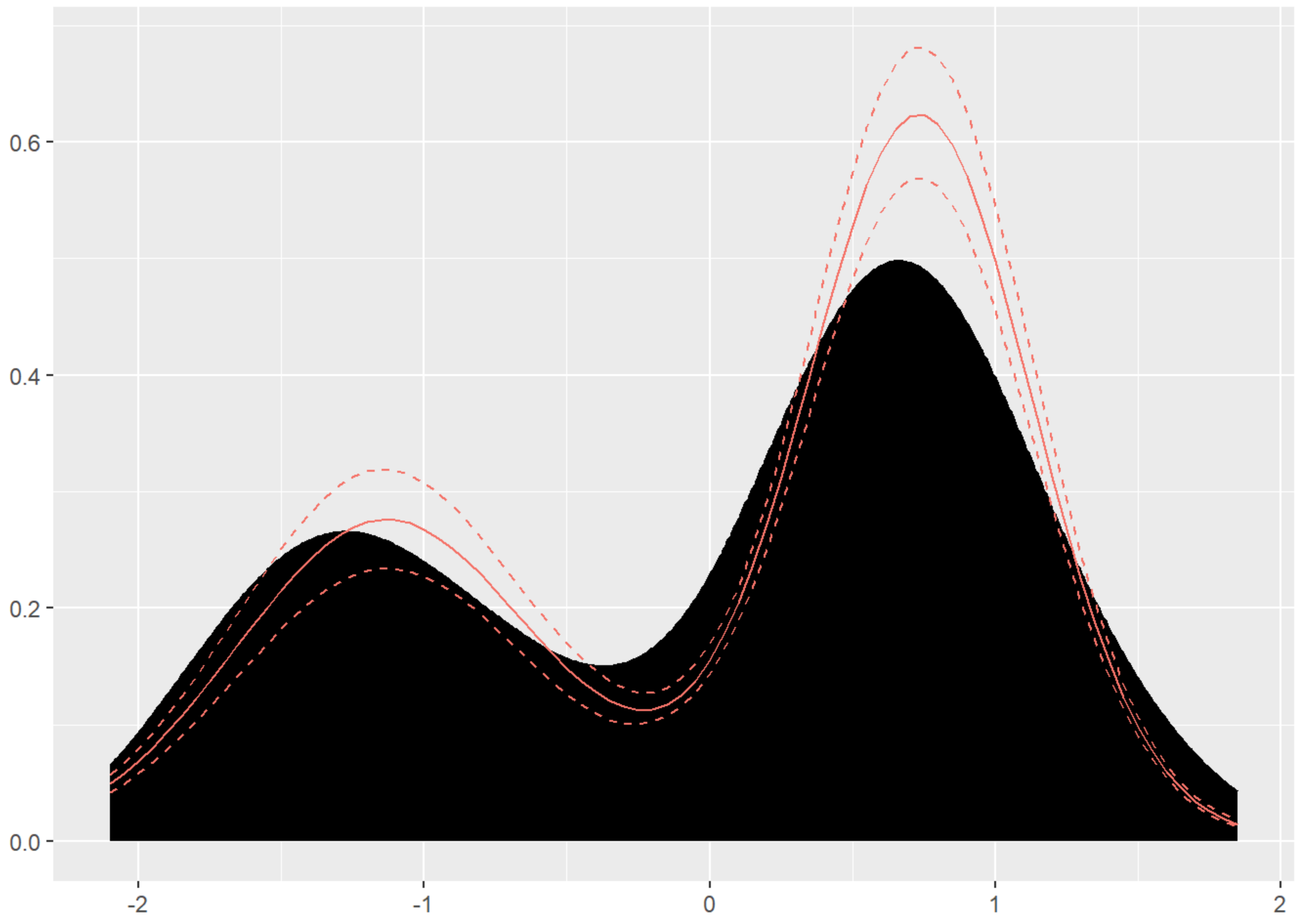
```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
set.seed(2)
```

Easy Test

Univariate Model for Faithful Dataset

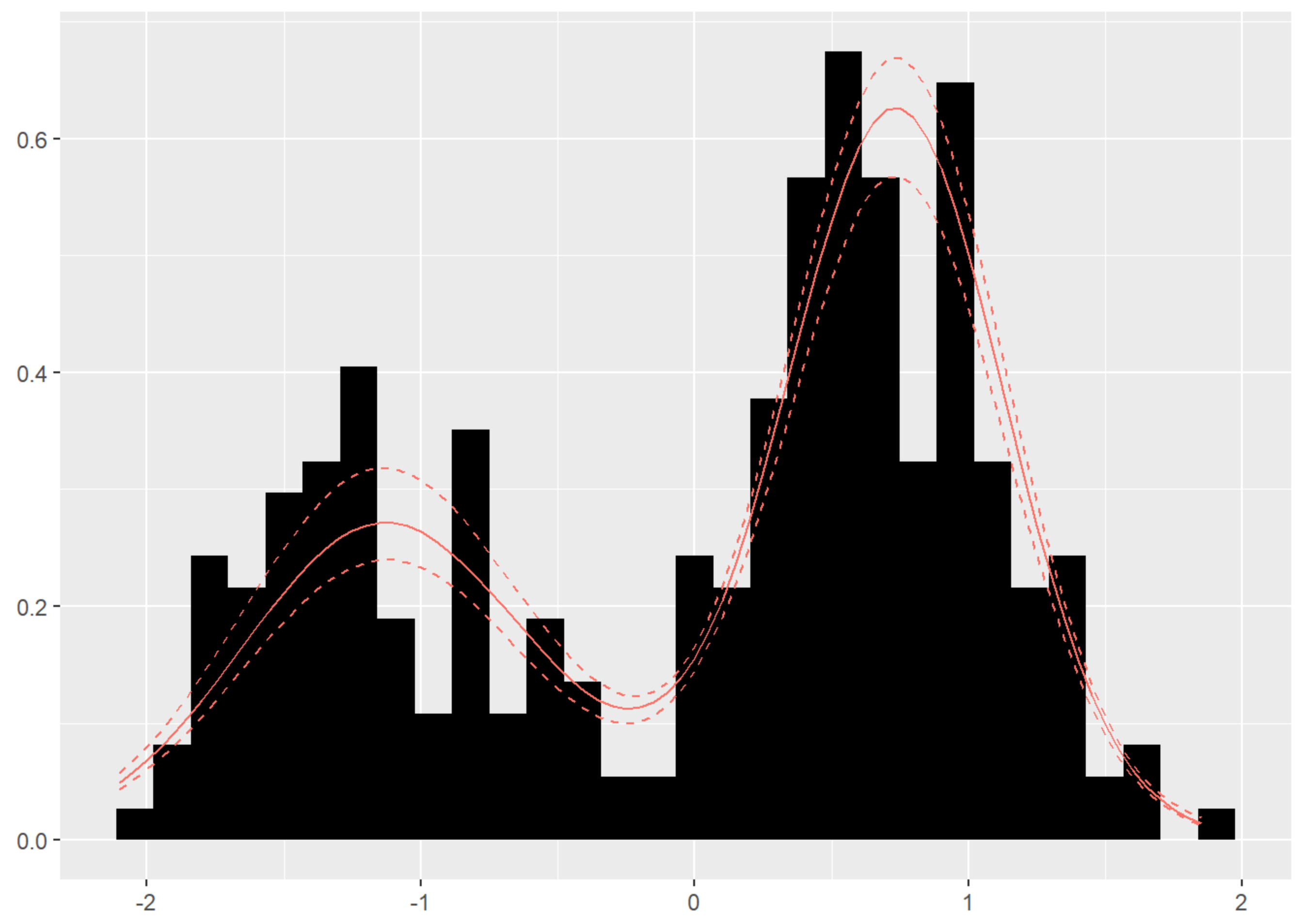
```
data("faithful")
its <- 500

# Standardize
df_faithful = scale(faithful$waiting)
dp_faithful <- DirichletProcessGaussian(df_faithful)
dp_faithful <- Fit(dp_faithful, its)
plot(dp_faithful)
```



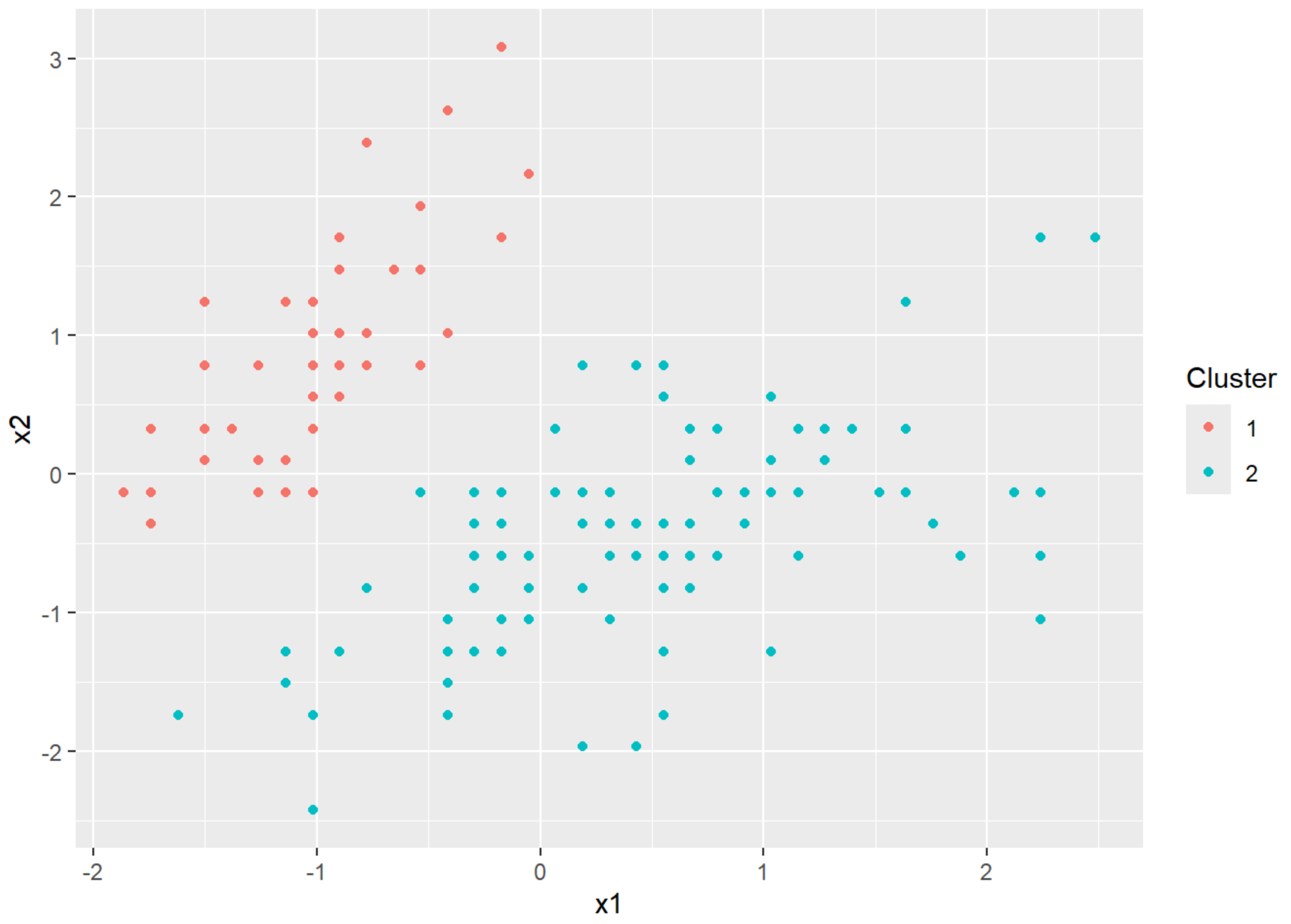
```
plot(dp_faithful, data_method="hist")
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## ℹ Please use `after_stat(density)` instead.
## ℹ The deprecated feature was likely used in the dirichletprocess package.
##   Please report the issue at
##   <https://github.com/dm13450/dirichletprocess/issues>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
data("iris")

# Standardize
df_iris = scale(iris[,1:4])
dp_iris <- DirichletProcessMvnormal(df_iris, numInitialClusters = nrow(iris))
dp_iris = Fit(dp_iris, 1000)
# Plot
plot(dp_iris)
```



##

Medium

- First Step: Generate random data from a lognormal distribution mixture model
- Second Step: Fit and simulate data from DP model
- Last Step : Sample data from posterior distribution,and report 95% CI

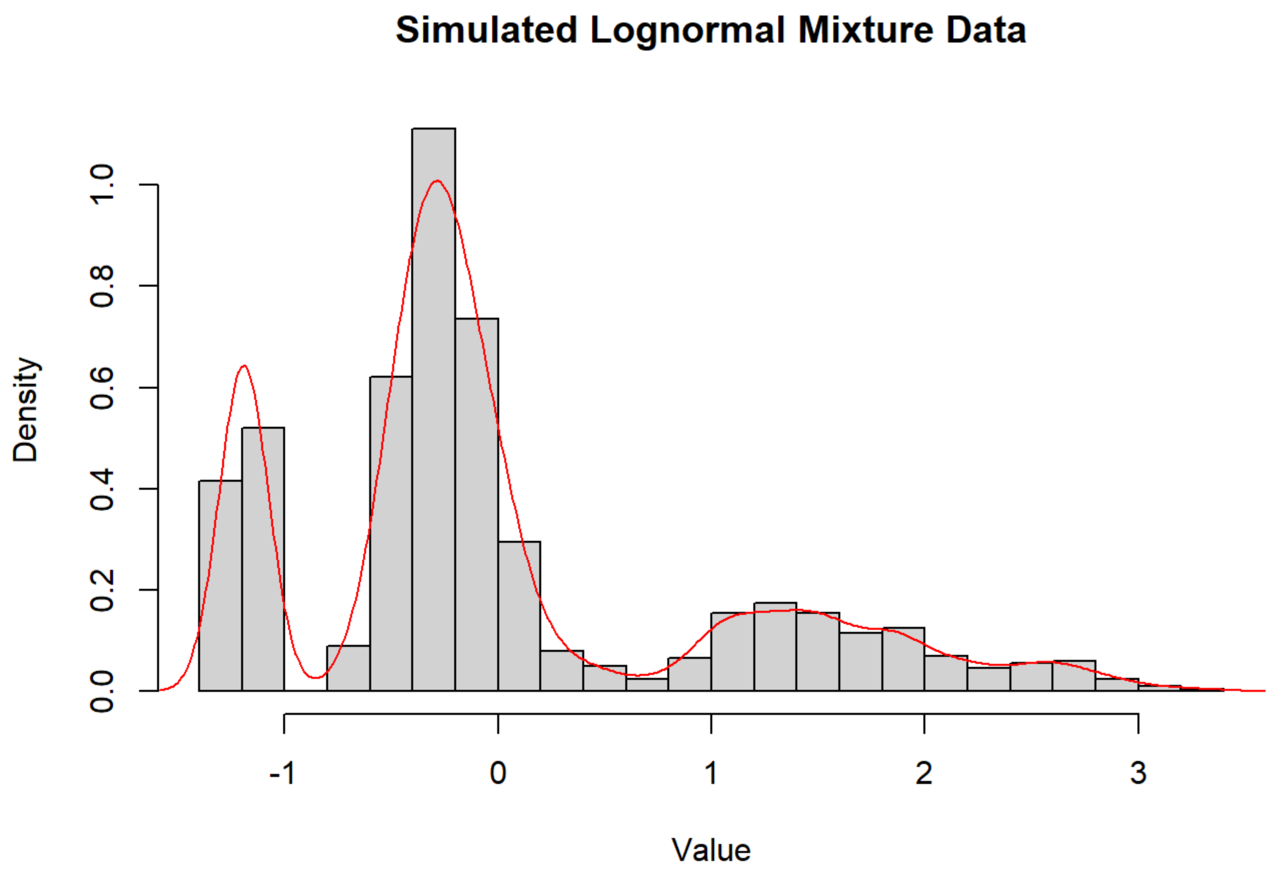
other tasks

- Explore hyperparameter in Prior: alpha and report its effect on the number of clusters.
- Explore different prior distribution choices effect on the alpha chain

$$y = 0.2lognormal(0, 0.2) + 0.6lognormal(2, 0.2) + 0.2lognormal(3, 0.2)$$

### Sample and simulate data

```r
# Sample from mixlognormal
n = 1000
component = sample(0:2, size = n, replace =TRUE, prob = c(0.2,0.6,0.2))
df_lnorm = numeric(n)
df_lnorm[component == 0] = rlnorm(sum(component == 0), meanlog = 0, sdlog = 0.2)
df_lnorm[component == 1] = rlnorm(sum(component == 1), meanlog = 2, sdlog = 0.2)
df_lnorm[component == 2] = rlnorm(sum(component == 2), meanlog = 3, sdlog = 0.2)

# Standardize data and record parameters
mean_original <- attr(scale(df_lnorm), "scaled:center")
sd_original <- attr(scale(df_lnorm), "scaled:scale")
df_lnorm = scale(df_lnorm)


hist(df_lnorm, breaks=30, probability=TRUE,
     main="Simulated Lognormal Mixture Data",
     xlab="Value")
lines(density(df_lnorm), col="red")
```



Compare number of clusters under different alpha priors

$$\alpha \sim Gamma(a, b)$$

with a is the shape parameter and b is the ratio parameter which our certainty towards alpha. We expect higher alpha generates more clusters.

```
# low alpha prior
dp_low_alpha <- DirichletProcessGaussian(df_lnorm, alphaPrior = c(1, 2))
dp_low_alpha <- Fit(dp_low_alpha, 2000)

# Default mod alpha prior
dp_mod_alpha <- DirichletProcessGaussian(df_lnorm, alphaPrior = c(2, 4))
dp_mod_alpha <- Fit(dp_mod_alpha, 2000)

# high alpha prior
dp_high_alpha <- DirichletProcessGaussian(df_lnorm, alphaPrior = c(5, 2))
dp_high_alpha <- Fit(dp_high_alpha, 2000)
```

```
cat("Number of clusters"," \n")
```

```
## Number of clusters
```

```
cat("low alpha:", length(unique(dp_low_alpha$clusterLabels)), "\n")
```

```
## low alpha: 4
```

```
cat("moderate alpha:", length(unique(dp_mod_alpha$clusterLabels)), "\n")
```

```
## moderate alpha: 5
```

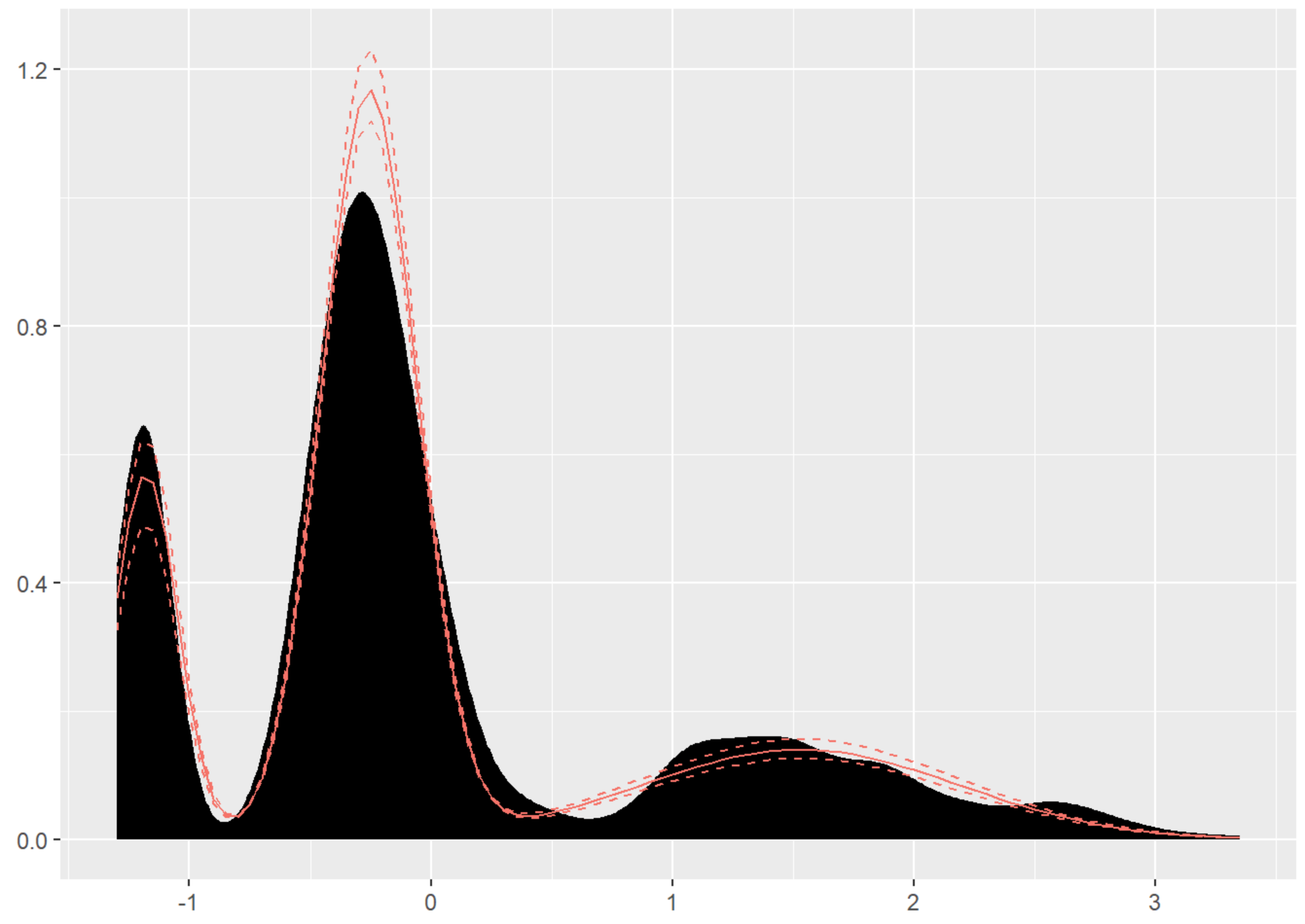
```
cat("high alpha:", length(unique(dp_high_alpha$clusterLabels)), "\n")
```

```
## high alpha: 12
```

Calculate 95% CI of posterior Distribution and Parameters

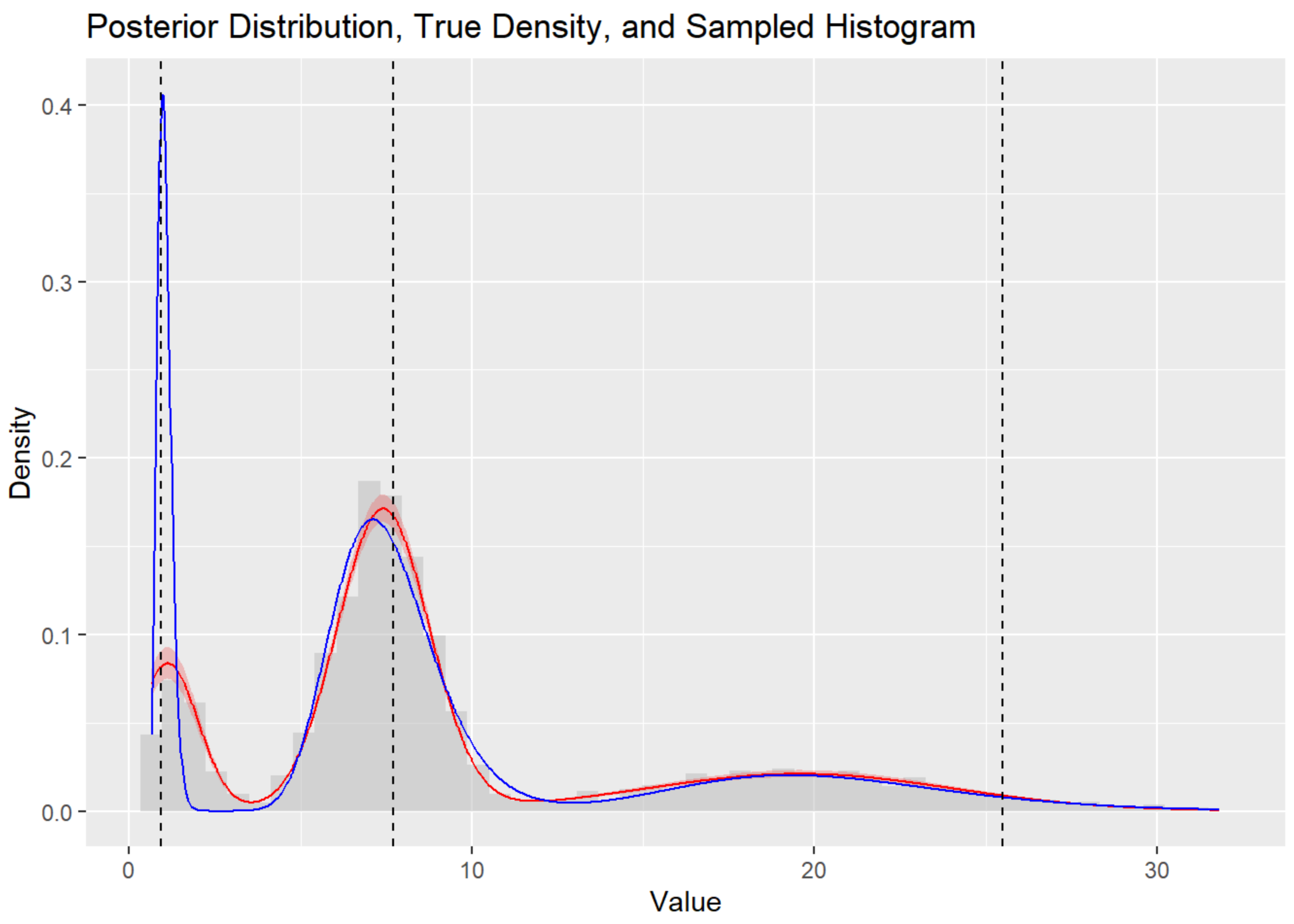
We calculate the 95% CI for posterior distribution under the low alpha prior set with $\alpha = 1, \beta = 2$

```
# hist(df_lnorm, breaks = 30)
plot(dp_low_alpha)
```



$$df_{lnorm} = \frac{df_{unscaled} - Mean}{Sd}$$

```r
x <- seq(min(df_lnorm), max(df_lnorm), length.out = 1000)

# Posterior frame
df_pos <- PosteriorFrame(dp_low_alpha, x, 1000)

# Unscale posterior x and density
df_pos$x <- df_pos$x * sd_original + mean_original
density_cols <- c("Mean", "X5.", "X95.")
df_pos[density_cols] <- lapply(df_pos[density_cols], `/`, sd_original)

# True density
trueFrame <- data.frame(
  x = df_pos$x,
  y = 0.2 * dlnorm(df_pos$x, meanlog = 0, sdlog = 0.2) +
      0.6 * dlnorm(df_pos$x, meanlog = 2, sdlog = 0.2) +
      0.2 * dlnorm(df_pos$x, meanlog = 3, sdlog = 0.2)
)

# Sample from Posterior Distribution
samples <- sample(df_pos$x, size = 5000, replace = TRUE, prob = df_pos$Mean)
quantiles <- quantile(samples, probs = c(0.025, 0.5, 0.975))

# Plot posterior and true density
ggplot() +
  geom_histogram(aes(x = samples, y = ..density..), bins = 50, fill = "gray", alpha = 0.5) +
  geom_ribbon(data = df_pos, aes(x = x, ymin = X5., ymax = X95.), fill = "red", alpha = 0.2) +
  geom_line(data = df_pos, aes(x = x, y = Mean), colour = "red") +
  geom_line(data = trueFrame, aes(x = x, y = y), colour = "blue") +
  geom_vline(xintercept = quantiles, linetype = "dashed", colour = "black") +
  labs(title = "Posterior Distribution, True Density, and Sampled Histogram",
       x = "Value",
       y = "Density")
```

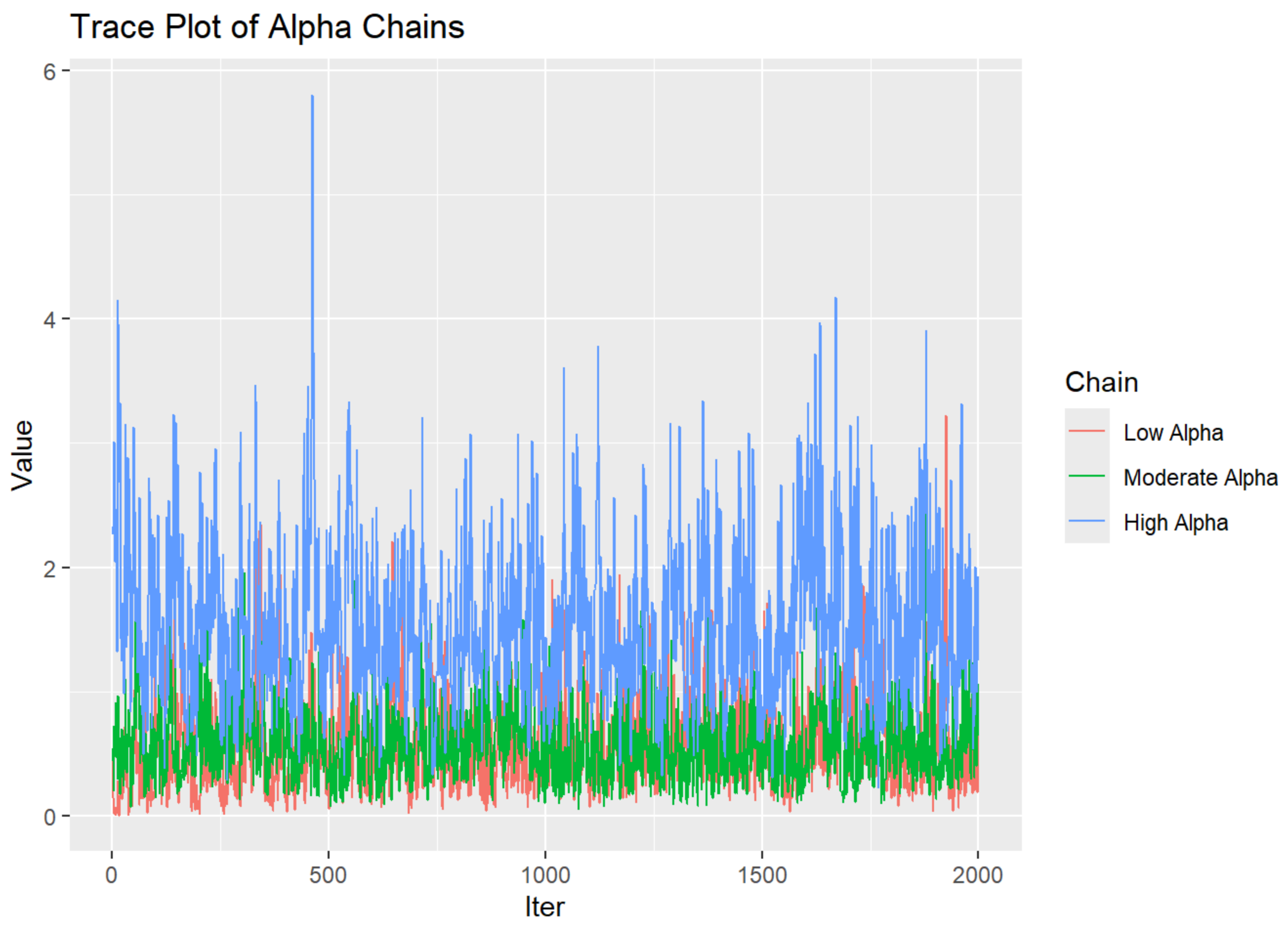


```
cat("Credential Interval of  Posterior Samples:","[",quantiles[0],",",quantiles[2],"]")
```

```
## Credential Interval of  Posterior Samples: [  , 7.717019 ]
```

Compare the alpha chain under different alpha priors

```
alphaFrame <- data.frame(Chain1 = dp_low_alpha$alphaChain, Chain2 = dp_mod_alpha$alphaChain, Chain
3 = dp_high_alpha$alphaChain, Iter=seq_len(2000))
alphaFrame %>% gather(Chain, Value, -Iter) -> alphaFrameTidy

ggplot(alphaFrameTidy, aes(x=Iter, y=Value, colour=Chain))+
  geom_line()+
  scale_color_discrete(labels = c("Low Alpha", "Moderate Alpha", "High Alpha")) +
  labs(title = "Trace Plot of Alpha Chains")
```



Hard

The dirichletprocess package currently implements Dirichlet process mixture models using Gaussian, Beta and Weibull kernels. Now we define a Dirichlet process mixture model (DPMM) in which each mixture component is a multivariate logistic distribution. The logistic distribution has heavier tails, which often increases the robustness of analyses based on it compared with using the normal distribution. By modelling data with a mixture of logistic components, the model can identify the data that do not fit in clusters well.

Model Set Up: DP with multivariate logistic distribution as its component

Suppose we have pbserved data: $y_1, y_2, \dots, y_n$ with $y_i \in R^d$

- Likelihood: $y_i \sim Logistic(\theta_i)$ where $\theta_i = (\mu_i, s_i)$
- Prior:$\theta_i \sim G, G \sim DP(\alpha, G_0)$

$$\mu \sim N_d(m_0, S_0) \quad and \quad s \sim Inverse - Gamma(a_0, b_0)$$

Since the logistic distribution is not conjugate to the usual choices of priors, we use MH-MC to update component parameters.

```r
# Likelihood function
Likelihood.mvlogistic <- function(mdObj, x, theta) {
  nComp <- dim(theta[[1]])[3]
  nObs <- nrow(x)
  d <- ncol(x)
  likelihoods <- matrix(0, nrow = nObs, ncol = nComp)

  for (i in 1:nComp) {
    mu_i <- theta[[1]][, , i]
    s_i <- theta[[2]][, , i]
    likelihoods[, i] <- apply(x, 1, function(row) {
      prod(exp(-(row - as.vector(mu_i)) / as.vector(s_i)) /
             (as.vector(s_i) * (1 + exp(-(row - as.vector(mu_i)) / as.vector(s_i)))^2))
    })
  }
  return(likelihoods)
}

# Prior draw function
PriorDraw.mvlogistic <- function(mdObj, n = 1) {
  priorParameters <- mdObj$priorParameters
  d <- length(priorParameters$mu0)

  # Draw mu: multinorm
  mu <- array(0, dim = c(d, 1, n))
  for (i in 1:n) {
    mu[, , i] <- matrix(mvtnorm::rmvnorm(1, mean = priorParameters$mu0, sigma = priorParameters$S
0), ncol = 1)
  }

  # Draw s: inverse gamma
  s <- array(0, dim = c(d, 1, n))
  for (i in 1:n) {
    s_vec <- numeric(d)
    for (j in 1:d) {
      s_vec[j] <- 1 / rgamma(1, shape = priorParameters$a0, rate = priorParameters$b0)
    }
    s[, , i] <- matrix(s_vec, ncol = 1)
  }

  theta <- list(mu = mu, s = s)
  return(theta)
}

# Posterior draw
PosteriorDraw.mvlogistic <- function(mdObj, x, n = 1, n_iter = 1000, burn_in = 500, ...) {
  priorParameters <- mdObj$priorParameters
  d <- ncol(x)

  # Initialize parameters
  # set mu to sample mean
  # Set s to sample standard deviation
```

```
current_mu <- colMeans(x)
current_s <- apply(x, 2, sd)
current_s[current_s == 0] <- 1
current_log_s <- log(current_s)

log_posterior <- function(mu, log_s) {
  s <- exp(log_s)
  # Compute log likelihood
  log_lik <- sum(apply(x, 1, function(row) {
    val <- sum( -(row - mu) / s - log(s) - 2 * log(1 + exp(-(row - mu) / s)) )
    if (!is.finite(val)) return(-Inf)
    return(val)
  }))

  # Priors, assume iid
  log_prior_mu <- sum(dnorm(mu, mean = priorParameters$mu0,
                            sd = sqrt(diag(priorParameters$S0)), log = TRUE))
  a0 <- priorParameters$a0; b0 <- priorParameters$b0
  log_prior_s <- sum(a0 * log(b0) - lgamma(a0) - a0 * log_s - b0 / exp(log_s))

  total <- log_lik + log_prior_mu + log_prior_s
  if (!is.finite(total)) total <- -Inf
  return(total)}

chain_mu <- matrix(0, nrow = n_iter, ncol = d)
chain_log_s <- matrix(0, nrow = n_iter, ncol = d)
chain_mu[1, ] <- current_mu
chain_log_s[1, ] <- current_log_s
current_lp <- log_posterior(current_mu, current_log_s)

# Proposal standard deviations
prop_sd_mu <- rep(0.1, d)
prop_sd_log_s <- rep(0.1, d)

for (iter in 2:n_iter) {
  # tuning parameters - adjust
  proposed_mu <- current_mu + rnorm(d, 0, prop_sd_mu)
  proposed_log_s <- current_log_s + rnorm(d, 0, prop_sd_log_s)
  proposed_lp <- log_posterior(proposed_mu, proposed_log_s)

  # reject proposals with infinite proposed_lp
  if (!is.finite(proposed_lp)) {
    accept <- FALSE
  } else {
    accept_prob <- exp(proposed_lp - current_lp)
    if (is.na(accept_prob)) accept_prob <- 0  # safeguard if NA
    accept <- runif(1) < accept_prob
  }

  if (accept) {
    current_mu <- proposed_mu
    current_log_s <- proposed_log_s
```

```r
      current_lp <- proposed_lp
    }

    chain_mu[iter, ] <- current_mu
    chain_log_s[iter, ] <- current_log_s
  }



  # Compute the posterior parameters: average of the post-burn-in
  final_mu <- colMeans(chain_mu[(burn_in + 1):n_iter, , drop = FALSE])
  final_s  <- exp(colMeans(chain_log_s[(burn_in + 1):n_iter, , drop = FALSE]))

  # Return the parameters 3D array
  mu_out <- array(matrix(final_mu, ncol = 1), dim = c(d, 1, n))
  s_out  <- array(matrix(final_s,  ncol = 1), dim = c(d, 1, n))
  return(list(mu = mu_out, s = s_out))
}

Predictive.mvlogistic <- function(mdObj, x, n_draws = 1000, ...) {
  theta_draws <- PosteriorDraw.mvlogistic(mdObj, x, n = n_draws, ...)
  lik <- Likelihood.mvlogistic(mdObj, x, theta_draws)
  pred <- rowMeans(lik)
  return(pred)
}

MvlogisticCreate <- function(priorParameters) {
  mdObj <- MixingDistribution("mvlogistic", priorParameters, "nonconjugate")
  return(mdObj)
}

DirichletProcessMvlogistic <- function(y,
                                       g0Priors,
                                       alphaPriors = c(2, 4),
                                       numInitialClusters = 1) {
  if (!is.matrix(y)) {
    y <- matrix(y, ncol = length(y))
  }

  # set defaults base priors
  if (missing(g0Priors)) {
    d <- ncol(y)
    g0Priors <- list(mu0 = rep(0, d),
                     S0 = diag(d),
                     a0 = 2,
                     b0 = 2)
  }

  mdobj <- MvlogisticCreate(g0Priors)
  dpobj <- DirichletProcessCreate(y, mdobj, alphaPriors)
  dpobj <- Initialise(dpobj, numInitialClusters = numInitialClusters)
  return(dpobj)
}
```

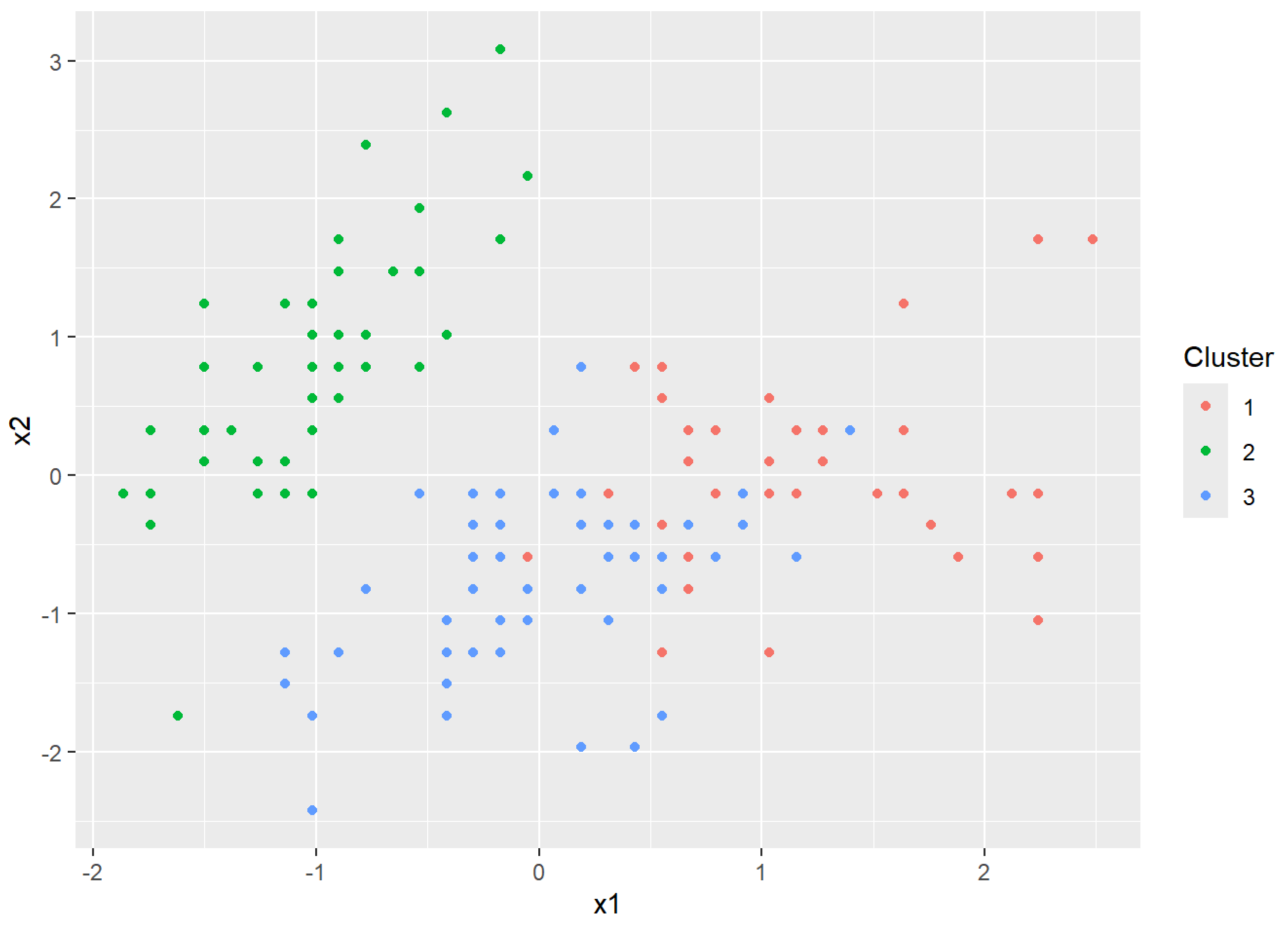
Model Testing

use iris as example:

```
data("iris")

# Standardize
df_iris = scale(iris[,1:4])
dp_iris <- DirichletProcessMvlogistic(df_iris, numInitialClusters = nrow(iris))
```

```
## Accept Ratio:  0.004
```

```
dp_iris = Fit(dp_iris, 1000)
# Plot
plot(dp_iris)
```



Reference

- http://127.0.0.1:25612/library/dirichletprocess/doc/dirichletprocess.pdf (http://127.0.0.1:25612/library/dirichletprocess/doc/dirichletprocess.pdf)

- https://cran.r-project.org/web/packages/dirichletprocess/dirichletprocess.pdf (https://cran.r-project.org/web/packages/dirichletprocess/dirichletprocess.pdf)